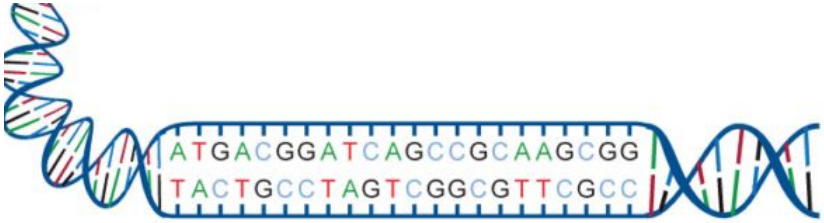
Processamento de Cadeias de Caracteres

Prof. José J. Camata
Prof.a. Bárbara Quintela
camata@ice.ufjf.br
barbara@ice.ufjf.br

Cadeias de Caracteres

- Uma cadeia é uma sequência linear de caracteres, tipicamente podendo ser muito longa
- Cadeias são centrais em sistemas de processamento de textos, filtro de spam, recuperação de informação, detecção de plágio, estudo de sequências de DNA em biologia computacional, etc.
- Essas cadeias podem ser bem grandes e algoritmos eficientes são necessários para manipulá-los.



Cadeias de Caracteres

- Abordaremos algumas questões relacionadas com cadeias de caracteres nos seguintes tópicos
  - **Casamento de padrões em cadeia de caracteres (String matching)**
  - **Codificação e Compressão**

Cadeias de Caracteres

- Notação:
  - T, usado para representar um texto, ou seja, uma sequência de caracteres ou letras.
  - P é um sequência de caracteres que se deseja procurar em T
  - |T| significa o comprimento de T
  - T_j é o caracter na posição *j.*
  - T[i,...,j] é um sequência de T que começa na posição i e termina em j.
  - Os primeiros caracteres do padrão P e o texto T estão na posição 1.
- Exemplo:
  - T = “ABACAABACC**ABACAB**AABB”
  - |T| = 20
  - P = “ABACAB” corresponde a trecho T[11,...,16]

Casamento de Padrão

- Uma operação fundamental sobre cadeias é o casamento de padrão:
  - consiste encontrar uma cópia exata do padrão P de tamanho m = |P| no texto T de tamanho n = |T|.
  - O problema de casamento de padrão pode ser visto também como um problema de busca com o padrão sendo a chave.
- Duas soluções serão estudadas:
  - **Força Bruta:** simples mas pouco eficiente. Tem complexidade O(nm)
  - **Algoritmo de Knuth, Morris e Pratt:** mais elaborado e com complexidade O(n).

Nota histórica

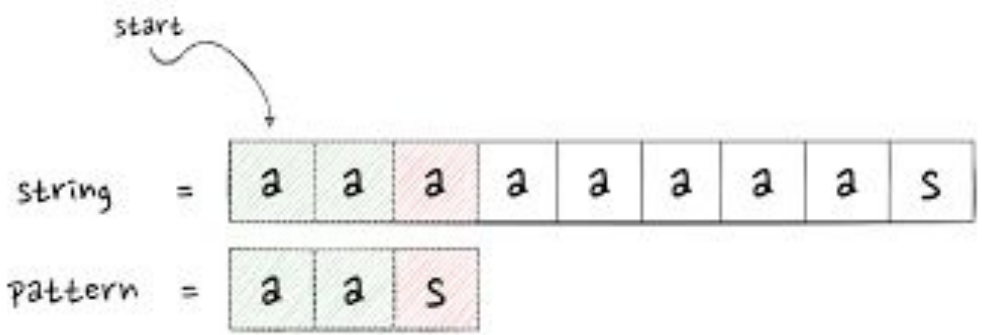
- Em 1970, S.A. Cook provou um resultado teórico sobre um tipo particular de autômato que implicava na existência de um algoritmo de casamento de padrão com tempo proporcional a M + N no pior caso.
- D. E. Knuth e V. R. Pratt seguindo a construção que Cook usara na demonstração do seu teorema obtiveram um algoritmo relativamente simples e prático.
- Ocorreu também que J. H. Morris descobriu praticamente o mesmo algoritmo como solução de um problema de edição de texto
- Os três cientistas, Knuth, Morris e Pratt, não se preocuparam em publicar o algoritmo até 1976

Nota histórica

- Nesse meio tempo, R. S. Boyer e J. S. Moore (e, independentemente, R. W. Gosper) descobriram um algoritmo que é muito mais rápido em muitas aplicações.
- Muitos editores de texto usam esse algoritmo para busca de cadeias.
- Em 1980, M. O. Rabin e R. M. Karp desenvolveram um algoritmo tão simples quanto o de força bruta que roda virtualmente sempre em tempo proporcional a M + N.
- Além disso, o algoritmo deles estende-se facilmente a padrões bidimensionais que o torna mais útil que os outros para processamento de figuras

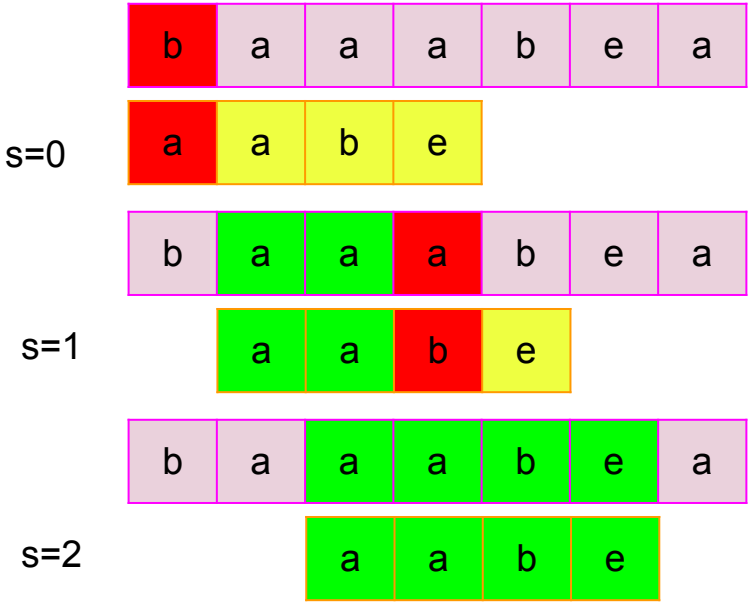
Algoritmo de Força Bruta

- O método óbvio para casamento de padrão resume-se em testar, em cada posição do texto onde o padrão pode casar, se ele de fato casa
- O procedimento `Naive-String-Matcher` a seguir busca dessa maneira a primeira ocorrência do padrão P no texto T



Algoritmo Força Bruta

```
Naive-String-Matcher(P,T)
n = |T|
m = |P|
for s = 0 to n-m
  if P[1...m] == T[s+1,...s+m]
        return s+1
return -1
```



Exemplo:

T	b	a	a	a	b	e	a
P	a	a	b	e			

Em s = 2, temos que P[1...4] casa com o trecho T[3...6]

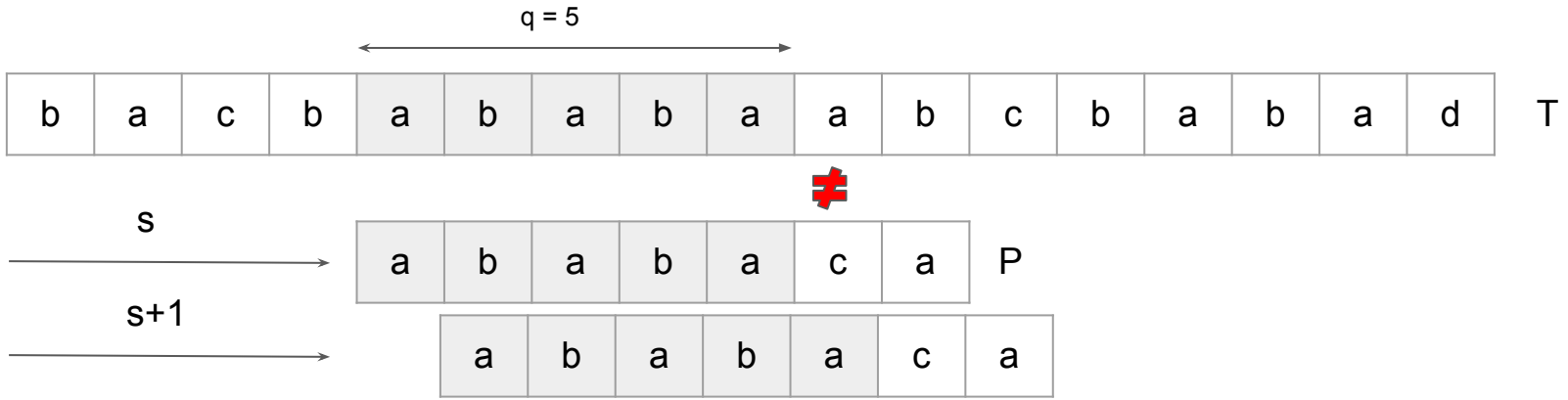
Algoritmo de Força Bruta

- Pior caso
  - O pior caso ocorre quando, por exemplo, o padrão e o texto são os dois uma sequência de zeros seguidos por um 1
    - Por exemplo, 00001 e 000000000000000001
  - Ou seja, quando é preciso percorrer praticamente todo P várias vezes a cada posição de T, estando P no fim da cadeia
  - Complexidade: m*n

Algoritmo Knuth-Morris-Pratt (KMP)

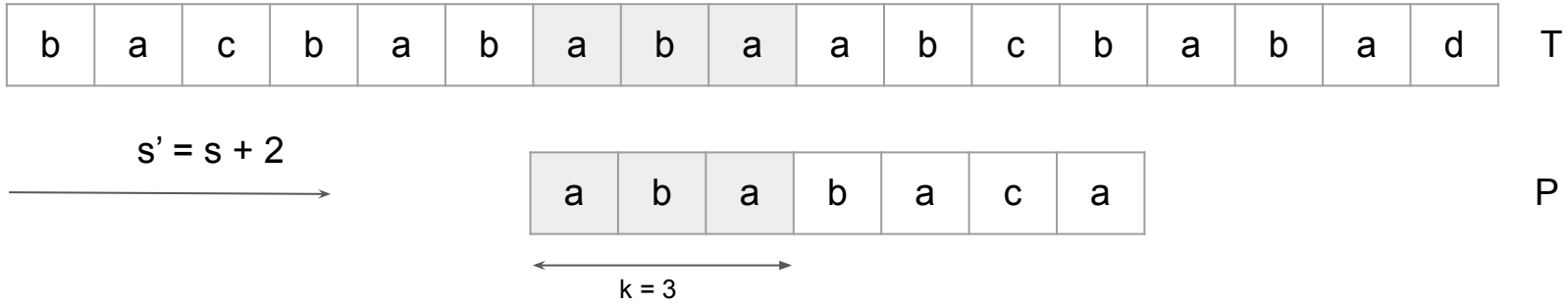
- Ideia básica:
  - No algoritmo de força bruta, quando ocorre uma diferença entre T[i] e P[j], <u>não seria melhor um deslocamento maior de P para a direita evitando comparações redundantes</u>?
  - Quando ocorre um descasamento, P tem em si a informação necessária para determinar onde começar a próxima comparação, ou seja, P inclui subsequências idênticas no início de P e antes do caractere incompatível.
- É o primeiro algoritmo cujo pior caso tem complexidade de tempo linear no tamanho do texto.

Algoritmo Knuth-Morris-Pratt (KMP) - Exemplo



Observe que o deslocamento s + 1 é necessariamente não válido, uma vez que o primeiro caractere do padrão estaria alinhado com um caractere do texto que sabe-se que não casará.

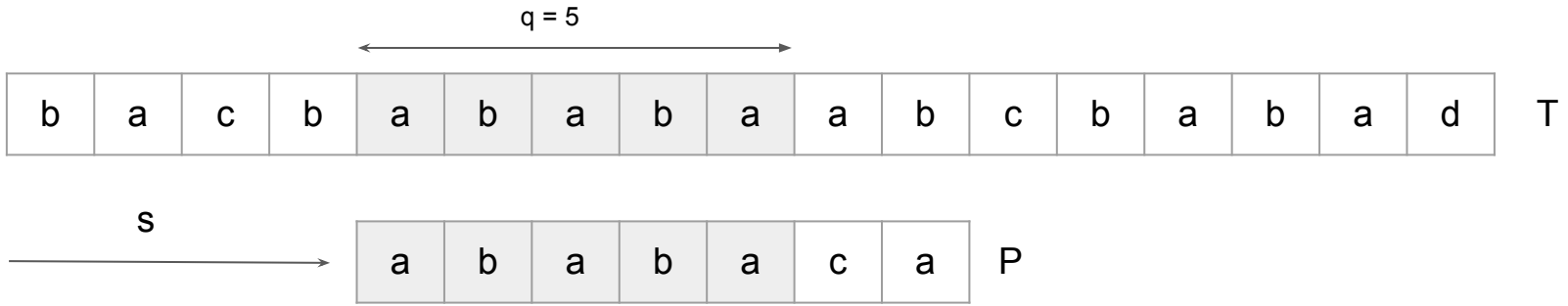
Algoritmo Knuth-Morris-Pratt (KMP) - Exemplo



O deslocamento s + 2 alinham os três primeiros caracteres do padrão P com três caracteres do texto que devem ser necessariamente correspondentes.

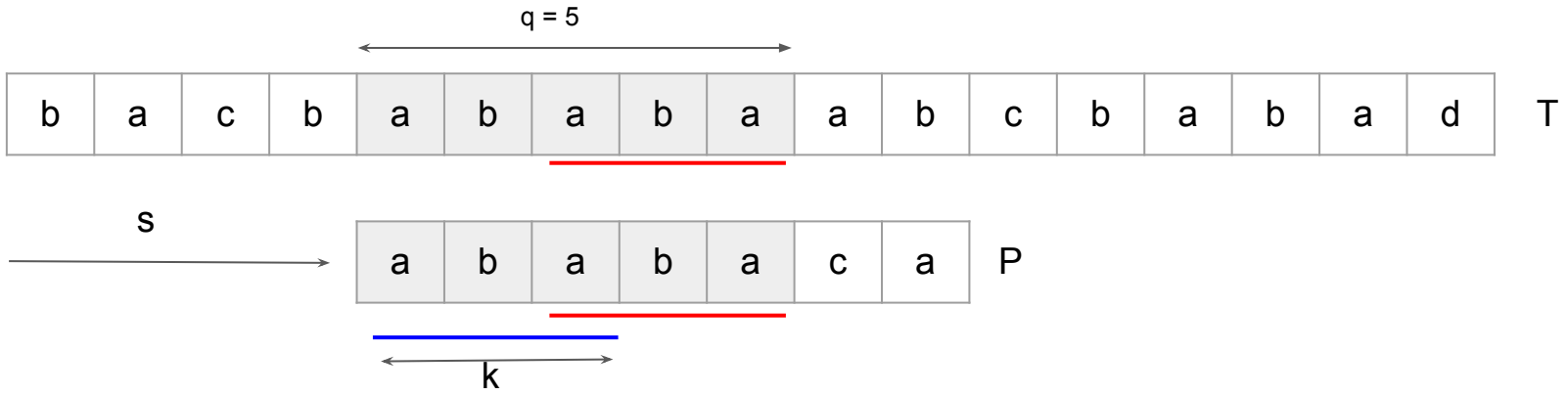
A questão então é: temos como saber qual é o deslocamento necessário para casar esses trechos e evitar comparações desnecessárias?

Algoritmo Knuth-Morris-Pratt (KMP)



Sabendo que P[1...q] está contido em T[s+1, s+q], queremos o prefixo próprio mais longo P[1...k] de P[1...q] que é também sufixo de T[s...q]. Note que, para encontrar o menor deslocamento s’ equivale a encontrar o maior prefixo k, pois s’ = s + (q - k).

Algoritmo Knuth-Morris-Pratt (KMP)



- Note que P[1...5] está contido em T[s+1, s+5] e que o **prefixo** próprio mais longo P[1...3] de P[1...5] é também **sufixo** de T[s+1...s+5].
- Para encontrar o menor deslocamento s' equivale a encontrar o maior prefixo k, pois s' = s + (q - k). Logo s' = 2.

Algoritmo Knuth-Morris-Pratt (KMP)

- Tem como saber antecipadamente quantas posições devem ser deslocadas?
- Para tal, utilizamos uma função chamada de **Função Prefixo, denotada por π.**

Função Prefixo

- A função prefixo encapsula o conhecimento sobre quantas posições deve-se caminhar para continuar procedendo o casamento do padrão, evitando comparações inúteis.
- Para tal, a função analisa o padrão fornecido e cria uma “tabela” de deslocamentos.
  - Para cada valor de q, armazena o número k de caracteres correspondentes do novo deslocamento s’
- Formalmente:

Dado um padrão P[1...m], a função prefixo para o padrão P é a função $\pi = [1,2,...,m] \rightarrow [0,1,...,m-1]$ tal que $\pi[q] = \max\{k : k < q \text{ e } p[1...k] \sqsupset P[1...q]$.

Função Prefixo

Exemplo do cálculo do vetor deslocamento:

i	1	2	3	4	5	6	7
P	a	b	a	b	a	c	a
π							

Função Prefixo

Exemplo do cálculo do vetor deslocamento:

i	1	2	3	4	5	6	7
P	a	b	a	b	a	c	a
π	0						

Função Prefixo

Exemplo do cálculo do vetor deslocamento:

i	1	2	3	4	5	6	7
P	a	b	a	b	a	c	a
π	0	0					

Função Prefixo

Exemplo do cálculo do vetor deslocamento:

i	1	2	3	4	5	6	7
P	a	b	a	b	a	c	a
π	0	0	1				

a	b	a

P[1...3]

Função Prefixo

Exemplo do cálculo do vetor deslocamento:

i	1	2	3	**4**	5	6	7
P	a	b	a	b	a	c	a
π	0	0	1	2			

a	b	a	b

P[1...4]

Função Prefixo

Exemplo do cálculo do vetor deslocamento:

i	1	2	3	4	**5**	6	7
P	a	b	a	b	a	c	a
π	0	0	1	2	3		

a	b	a	b	a	P[1...5]

Função Prefixo

Exemplo do cálculo do vetor deslocamento:

i	1	2	3	4	5	6	7
P	a	b	a	b	a	c	a
π	0	0	1	2	3	0	

a	b	a	b	a	c	P[1...6]

Função Prefixo

Exemplo do cálculo do vetor deslocamento:

i	1	2	3	4	5	6	7
P	a	b	a	b	a	c	a
π	0	0	1	2	3	0	1

a	b	a	b	a	c	a	P[1...6]

$\pi[q] = \max\{k : k < q \text{ e } p[1...k] \sqsupset P[1...q]$

Pseudocódigo KMP

```
KMP-Matcher(T, P)
Entrada: Cadeias de caracteres T e P
n ← |T|;
m ← |P|;
π ← ComputaFuncaoPrefixo(P);
q ← 0;                                   // Número de caracteres correspondentes
// considere i ≤ n
para i ← 1 até n faça                    // Varre o texto da esquerda para direita
    enquanto q > 0 e P[q + 1] ≠ T[i] faça
        q ← π[q];                        // próximo caractere não correspondente
    fim
    se P[q + 1] = T[i] então
        q ← q + 1;                       // próximo caractere é correspondente
    fim
    se q = m então                       // P inteiro é correspondente?
        Imprima "O padrão ocorre com deslocamento i − m";
        q ← π[q];                        // Procura próxima correspondência.
    fim
fim
```

Pseudocódigo Função Prefixo

```
ComputaFuncaoPrefixo(P)
Entrada: Cadeia de caracteres P
m ← |P|;
Crie π[1..m]; é um arranjo completamente novo
π[1] ← 0;
k ← 0;
// considere q ≤ m
para q ← 2 até m faça
    enquanto k > 0 e P[k + 1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k + 1] = P[q] então
        k ← k + 1;
    fim
    π[q] ← k;
fim
retorna (π);
```

Exercício

Calcule a função prefixo para o padrão abaixo

a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b

Exercício

Calcule a função prefixo para o padrão abaixo

P	a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π																			

1 **ComputaFuncaoPrefixo**(P)
Entrada: Cadeia de caracteres P
2 $m \leftarrow |P|$;
3 **Crie** $\pi[1..m]$; é um arranjo completamente novo
4 $\pi[1] \leftarrow 0$;
5 $k \leftarrow 0$;

Calcule a função prefixo para o padrão abaixo

k

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P	a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π	0																		

m=19

1 **ComputaFuncaoPrefixo**(P)
Entrada: Cadeia de caracteres P
2 $m \leftarrow |P|$;
3 **Crie** $\pi[1..m]$; é um arranjo completamente novo
4 $\pi[1] \leftarrow 0$;
5 $k \leftarrow 0$;

Calcule a função prefixo para o padrão P abaixo:

k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π	0																			

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k == 0 (F)

P[1] == P[2] (F)

π[2] ←0

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0																	

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k==0 (F)

P[1] == P[3] (V) Logo k ←1

π[3] ←1

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1																

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k>0 e P[2] != P[4] (F)

P[2] == P[4] (V) Logo k ←2

π[4] ←2

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2															

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k > 0 e P[3] != P[5], logo k ←π[2] (0)

P[1] == P[5] (F)

π[5] ←0

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0														

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k > 0 (F)

P[1] == P[6] (V) Logo k ←1

π[6] ←1

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1													

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k >0 e P[2] != P[7] (F)

P[1] == P[6] (V) Logo k ←2

π[7] ←2

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2												

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k>0 e P[2] != P[8], logo k ←π[2] (0)

P[1] == P[8] (F)

π[0] ←0

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0											

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k > 0 (F)

P[1] == P[9] (V) logo k ←1

π[9] ←1

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1										

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k>0 e P[2] != P[10] (F)

P[2] == P[10] (V), k ←2

π[10] ←2

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1	2									

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k > 0 e P[3] !=P[11] (V) logo k ←π[2] (0)

P[1] != P[11]!

π[11] ←0

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1	2	0								

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k > 0 (F)

P[1] == P[12] (V), logo k ←1

π[12] ←1

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π	0	0	1	2	0	1	2	0	1	2	0	1								

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k == 0, k > 0(F)

P[1] == P[12] (V)!, logo k ←1

π[12] ←1

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1	2	0	1							

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k > 0 e P[2] != P[13]! (F)

P[2] == P[13] (V), logo k ←2

π[13] ←2

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1	2	0	1	2						

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k > 0 e P[3] != P[14] (F)

P[3] == P[14] (V), logo k ←3

π[14] ←3

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1	2	0	1	2	3					

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k ==3, P[4] != P[15] (F)!

P[4] == P[15] (V) k ←4

π[15] ←4

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1	2	0	1	2	3	4				

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k> 0 e P[5] != P[16] (F)

P[5] == P[16] (V)! logo k ←5

π[16] ←5

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1	2	0	1	2	3	4	5			

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k > 0 e P[6] != P[17] (F)!

P[6] == P[17] (V) logo k ←6

π[17] ←6

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1	2	0	1	2	3	4	5	6		

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k>0 e P[7] != P[18] (F)!

P[7] == P[18] (V) logo k ←7

π[18] ←7

Calcule a função prefixo para o padrão

k k q

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1	2	0	1	2	3	4	5	6	7	

m=19

```
para q ← 2 até m faça
    enquanto k > 0 e P[k+1] ≠ P[q] faça
        k ← π[k];
    fim
    se P[k+1] = P[q] então
        k ← k+1;
    fim
    π[q] ← k;
fim
```

k > 0 e P[8] != P[19] (F)!

P[8] == P[19] (V) logo k ←8

π[19] ←8

Calcule a função prefixo para o padrão

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
P		a	b	a	b	b	a	b	b	a	b	b	a	b	a	b	b	a	b	b
π		0	0	1	2	0	1	2	0	1	2	0	1	2	3	4	5	6	7	8

m=19

Exercício 2:

1. Compute a função prefixo para P = abarba;
2. Use o algoritmo KMP para buscar P em T=abacaabaccabacabaabb.
3. Quantas comparações o método faz?

Visualizador:
https://cmps-people.ok.ubc.ca/ylucet/DS/KnuthMorrisPratt.html

Processamento de Cadeias de Caracteres

Prof. José J. Camata
camata@ice.ufjf.br